HOMEWORK #1:

Going Fishing

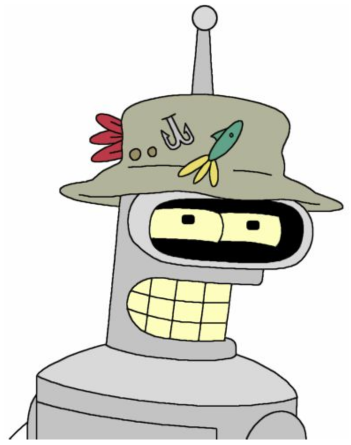
Due Date: Monday, September the 2nd, 11:59pm

For this assignment, you will submit a single C++ file called ‘`fishbomb.cpp`’.
Remember to put your name and section at the top of your program file.
Your program should expect all input to come from ‘`cin`’, and all your output should be to ‘`cout`’.

Problem

The Planet Express crew are going fishing! and Bender is doing so the way robots are meant to do it: with dynamite!. Using sophisticated robot technology, Bender is scanning the surrounding area with a sonar device. The sonar’s data represents the area around Bender as a grid, with the number of fish in each cell of the grid.

Help Bender choose which coordinate of the sonar data grid to throw his dynamite stick in order to maximize the number of fish ~~blown~~ caught. Bender’s dynamite stick will blow an area of **P** by **P**, where **P** is the power level of the dynamite stick.
(**P** will always be a positive odd number, and Bender **can not** throw the dynamite in a location where the explosion will go beyond the boundaries of the data grid)



Ahhh… the joys of fishing

Input

The first line of the input gives the number of data grids **T**.
The first line of each test case contains the numbers **W, H**, and **P;** the width and height of the sonar data grid and the power of the dynamite stick. **H** rows of **W** data points follow, describing the number of fish detected by the sonar. **P** will always be less than or equal **H** and **W**.

Output

For each data grid, output one line containing "*#g: (x, y) f*", where ***g*** is the grid number (starting from 0), ***(x, y)*** is the cell coordinate (0,0 being the top-left corner) in the fishing area where Bender will get the most fish, and ***f*** is the number of fish to be catched.

Implementation Requirements

Given that you do not know beforehand how large a sonar data grid is, your program should dynamically allocate a 2D Array after the width and height of a grid is read. Make sure to de-allocate the 2D Array after you find the answer and before your program moves on to process the next grid.

Sample

Input	Output
`3`	`#0: (1, 2) 2`
`3 3 1`	`#1: (3, 1) 40`
`1 0 0`	`#2: (2, 1) 12`
`0 1 0`	
`0 2 1`	
`6 3 3`	
`4 1 2 0 0 6`	
`0 7 5 9 8 3`	
`4 4 6 2 8 1`	
`4 4 3`	
`1 0 0 3`	
`0 1 0 1`	
`0 2 1 4`	
`0 0 0 1`	